**Create reference data**: create a dataset that defines the set of permissible values your data may contain. For example, in a country data field, specify the list of country codes allowed.

**Extract data from different sources**: the basis for the success of subsequent ETL steps is to extract data correctly. Take data from a range of sources, such as non/relational databases, XML, JSON, CSV files, and convert it into a single format for standardized processing.

**Validate data**: Keep data that have values in the expected ranges and reject any that do not. For example, if you only want dates from the last year, reject any values older than 12 months. Analyze rejected records, on an on-going basis, to identify issues, correct the source data, and modify the extraction process to resolve the problem in future batches.

**Transform data**: Remove duplicate data (cleaning), apply business rules, check data integrity (ensure that data has not been corrupted or lost), and create aggregates as necessary. For example, if you want to analyze revenue, you can summarize the dollar amount of invoices into a daily or monthly total. You need to program numerous functions to transform the data automatically.

**Stage data**: You do not typically load transformed data directly into the target data warehouse. Instead, data first enters a staging database which makes it easier to roll back if something goes wrong. At this point, you can also generate audit reports for regulatory compliance, or diagnose and repair data problems.

**Publish to your data warehouse**: Load data to the target tables. Some data warehouses overwrite existing information whenever the ETL pipeline loads a new batch - this might happen daily, weekly, or monthly. In other cases, the ETL workflow can add data without overwriting, including a timestamp to indicate it is new. You must do this carefully to prevent the data warehouse from “exploding” due to disk space and performance limitations.

1. We pulled 3 csv files from Kaggle.com (competitions.csv, teams.csv, and results.csv).
2. Using Jupyter Notebook to clean, filter, join, and aggregate data covering BBQ competitions held in Kansas City, MO, we were able to merge three distinct csv files into one cohesive data set. The first issue was figuring out how the three data sets related to one another. Once we had our keys to the data, we were able to merge the sets together.

First, we had to manipulate the columns in order to get rid of duplicate names. Then we had to join the teams data with the results data through the team\_id key. Once we had our joined data, we were able to bring in the competition data using competition\_id merged with competition\_id in our previous joined data set.

After we got all our data adjusted and joined, we were able to clean the data set by establishing which columns were necessary and which structure we wanted. From there we simply staged the data in

1. We will place the final data in PostgreSQL under the name Kansas\_City\_BBQ.
2. The data collected includes all of the results from barbeques in Kansas City, MO, the BBQ competitions that took place in Kansas City, MO, all of the teams that participated in those competitions in Kansas City, MO, and the results from those competitions in Kansas City, MO.